

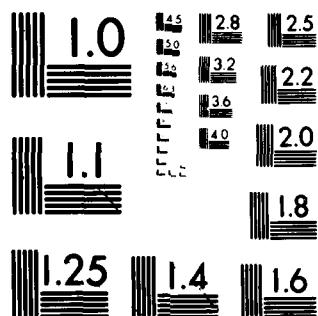
TRW DEFENSE AND SPACE SYSTEMS GROUP REDONDO BEACH CA F/6 9/2  
AIRBORNE SYSTEMS SOFTWARE ACQUISITION ENGINEERING GUIDEBOOK FOR--ETC(U)  
MAR 80 M BARRY F33657-76-C-0677  
TRW-30323-6011-TU-00 ASD-TR-80-5023 NL

UNCLASSIFIED

ASD-TR-80-5023

NL

END  
DATE  
FILMED  
2-80  
DTIC



MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

**LEVEL II**

ASD-TR-80-5023 ✓

①

**Airborne Systems**

**Software Acquisition Engineering Guidebook  
for  
SOFTWARE TESTING  
AND EVALUATION**

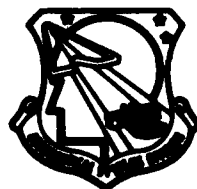
AD A090964

**MARCH 1980**

APPROVED FOR PUBLIC RELEASE;  
DISTRIBUTION UNLIMITED

DTIC  
ELECTE  
OCT 30 1980  
S D E

PREPARED FOR  
DEPUTY FOR ENGINEERING  
AERONAUTICAL SYSTEMS DIVISION  
WRIGHT-PATTERSON AFB, OH 45433



80 10 28 046

PREPARED BY  
TRW DEFENSE AND SPACE SYSTEMS GROUP  
ONE SPACE PARK  
REDONDO BEACH, CA 90278

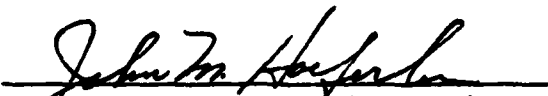
DDC FILE COPY

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

  
JOHN M. HOEFERLIN, Project Engineer  
Information Engineering Division

  
RICHARD J. SYLVESTER  
ASD Computer Resources Focal Point

FOR THE COMMANDER

  
ROBERT P. LAVOIE, Colonel, USAF  
Director of Avionics Engineering  
Deputy for Engineering

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify ASD/PA/ATA, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
	AD-A090964	
4. TITLE (and Subtitle)	5. TYPE OF REPORT & PERIOD COVERED	
Airborne Systems Software Acquisition Engineering Guidebook for Software Testing and Evaluation.	Final Repts.	
7. AUTHOR(s)	8. PERFORMING ORG. REPORT NUMBER	
M. Barry	TRW-30323-6011-TU-00	
	9. CONTRACT OR GRANT NUMBER(s)	
	F33657-76-C-0677	
9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
TRW Defense and Space Systems Group One Space Park Redondo Beach, CA 90278	PE 64740F Project 2238	
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE	
HQ ASD/ENAI Wright-Patterson AFB, Ohio 45433	March 1980	
	13. NUMBER OF PAGES	
	78	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report)	
ASD	UNCLASSIFIED	
TR-80-5023	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) Approved for Public Release, Distribution Unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Bottom Up/Top Down Testing, Computer Program Development Plan (CPDP), Configuration Management (CM), Development Test and Evaluation (DT&E), Discrepancy Report, Formal Qualification Test (FQT), Preliminary Qualification Test (PQT), Program Office (PO), Quality Assurance (QA), Retesting, Software Integration Tests, Software Life Cycle, Test Plan, Test Procedure.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This guidebook assists Air Force Program Office engineering and management personnel in interpreting and applying the principles of Software Testing and Evaluation to the acquisition of airborne systems software. The guidebook describes the planning and testing activities necessary to insure successful software testing. It presents checklists and references to supplement information in government documents and to summarize data from professional journals and books.		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 68 IS OBSOLETE

UNCLASSIFIED  
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409637

**UNCLASSIFIED**

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

**Block 19 Continued**

**Test Report, Test Tools, Unit Tests.**

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DOC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/_____	
Availability Codes	
Dist.	Avail and/or special
A	

**UNCLASSIFIED**

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

## PREFACE

This guidebook is one of a series of guidebooks intended to assist Air Force Program Office and Engineering personnel in software acquisition engineering for airborne systems. The contents of the guidebooks will be revised periodically to reflect changes in software acquisition policies and practices and feedback from users.

This guidebook was prepared under the direction of the Aeronautical Systems Division, Deputy for Engineering (ASD/EN) in coordination with the Space Division, Deputy for Acquisition Management (SD/AQM).

The entire series of Software Acquisition Engineering Guidebooks (Airborne Systems) is listed below along with ASD Technical Report numbers and NTIS accession numbers where available.

Regulations, Specifications and Standards	ASD-TR-78-6	ADA058428
Reviews and Audits	ASD-TR-78-7	ADA058429
Software Quality Assurance	ASD-TR-78-8	ADA059068
Configuration Management	ASD-TR-79-5024	ADA076542
Computer Program Documentation Requirements	ASD-TR-79-5025	ADA076543
Statements of Work and Requests for Proposal	ASD-TR-79-5026	ADA076544
Requirements Analysis and Specification	ASD-TR-79-5027	
Verification, Validation and Certification	ASD-TR-79-5028	
Microprocessors and Firmware	ASD-TR-80-5021	
Software Development Planning and Control	ASD-TR-80-5022	
Software Testing and Evaluation	ASD-TR-80-5023	
* Contracting for Software Acquisition	ASD-TR-80-5024	

* Software Cost Analysis and Estimating	ASD-TR-80-5025
* Supportable Airborne Software	ASD-TR-80-5026
* Software Development and Support Facilities	ASD-TR-80-5027
* SAE Guidebooks - Application and Use	ASD-TR-80-5028

---

\* These Guidebooks Available Fall 1980.



## CONTENTS

<b>PREFACE</b> .....	iii
<b>ABBREVIATIONS AND ACRONYMS</b> .....	ix
<b>1. INTRODUCTION</b> .....	1
1.1 Purpose and Scope of Software Testing and Evaluation .....	1
1.2 Life Cycle Relationship .....	2
1.3 Relationship to Other Guidebooks .....	2
1.4 Relationship to Verification, Validation and Certification .....	3
1.5 Contents of This Guidebook .....	4
1.5.1 Section 1: Introduction .....	4
1.5.2 Section 2: Relevant Documents .....	4
1.5.3 Section 3: General Guidance for Software Test and Evaluation .....	4
1.5.4 Section 4: Specific Guidance for Software Test and Evaluation .....	4
1.5.5 Appendix A .....	4
<b>2. RELEVANT DOCUMENTS</b> .....	5
2.1 Regulations, Specifications and Standards .....	5
<b>3. GENERAL GUIDELINES FOR SOFTWARE TEST AND EVALUATION</b> .....	7
3.1 Preliminary Concepts .....	7
3.1.1 Program Office .....	7
3.1.2 Contractor .....	8
3.1.3 Testing in the Software Life Cycle .....	10
3.2 Planning Considerations .....	16
3.2.1 Computer Program Development Plan .....	16
3.3 Test Conduct and Control .....	17
3.3.1 Test Design .....	17

## **CONTENTS (Concluded)**

3.3.2	Control of Software Tests .....	19
3.3.3	Control of the Software Products .....	22
3.4	Test Tools .....	26
3.5	Management Considerations .....	30
3.5.1	Organization .....	30
3.5.2	Resources and Facilities .....	35
3.5.3	Schedule .....	35
4.	SPECIFIC GUIDANCE FOR SOFTWARE TEST AND EVALUATION .....	37
4.1	Test Engineering and Specification .....	37
4.2	Software Testing and Evaluation .....	42
4.2.1	Unit Tests .....	42
4.2.2	Software Integration Tests .....	46
4.2.3	Preliminary Qualification Tests .....	53
4.2.4	Formal Qualification Test .....	53
4.3	Retesting and Modifications to Support System Testing .....	59
	APPENDIX A: ANNOTATED BIBLIOGRAPHY .....	61

## TABLES

3-1.	Software Related Testing Documents Reviewed at Reviews and Audits . . . . .	9
3-2.	Test Documentation . . . . .	12
3-3.	Items in the CPDP Related to Software Testing . . . . .	18
3-4.	Advantages and Disadvantages of Bottom Up and Top Down Development and Testing . . . . .	20
3-5.	Candidates for Control by Configuration Management . . . .	23
3-6.	Information on Discrepancy Reports . . . . .	24
3-7.	Information on Discrepancy Report Log . . . . .	25
3-8.	Advantages and Disadvantages of Development of Test Tools by Testers or Another Organization . . . . .	28
3-9.	Test Tools . . . . .	31
4-1.	CPDP Review Checklist, Testing Sections . . . . .	38
4-2.	Part 1 CPCI Development Specification Review Checklist . . . . .	41
4-3.	Unit Testing Review Checklist . . . . .	47
4-4.	CPCI Test Plan Review Checklist . . . . .	55
4-5.	CPCI Test Procedure Review Checklist . . . . .	57

## ILLUSTRATIONS

3-1.	Idealized Software Life Cycle . . . . .	11
4-1.	Sample UDF Cover Sheet . . . . .	44
4-2.	Sample TDF Cover Sheet . . . . .	52

## ABBREVIATIONS AND ACRONYMS

<b>AFLC</b>	<b>Air Force Logistics Command</b>
<b>AFSC</b>	<b>Air Force Systems Command</b>
<b>AFTEC</b>	<b>Air Force Test and Evaluation Center</b>
<b>AFR</b>	<b>Air Force Regulation</b>
<b>AQM</b>	<b>Acquisition Management Support</b>
<b>ASD</b>	<b>Aeronautical Systems Division</b>
<b>ATE</b>	<b>Automatic Test Equipment</b>
<b>CCB</b>	<b>Configuration Control Board</b>
<b>CDR</b>	<b>Critical Design Review</b>
<b>CDRL</b>	<b>Contract Data Requirements List</b>
<b>CFE</b>	<b>Contractor Furnished Equipment</b>
<b>CI</b>	<b>Configuration Item</b>
<b>CM</b>	<b>Configuration Management</b>
<b>CMC</b>	<b>Configuration Management Organization</b>
<b>CPC</b>	<b>Computer Program Component</b>
<b>CPCI</b>	<b>Computer Program Configuration Item</b>
<b>CPDP</b>	<b>Computer Program Development Plan</b>
<b>CRISP</b>	<b>Computer Resources Integrated Support Plan</b>
<b>CRWG</b>	<b>Computer Resources Working Group</b>
<b>DID</b>	<b>Data Item Description</b>
<b>DODD</b>	<b>Department of Defense Directive</b>
<b>DR</b>	<b>Discrepancy Report</b>
<b>DRL</b>	<b>Discrepancy Report Log</b>
<b>DT&amp;E</b>	<b>Development Test and Evaluation</b>

## ABBREVIATIONS AND ACRONYMS (Continued)

EN	Deputy for Engineering
FCA	Functional Configuration Audit
FOT&E	Follow-on Operational Test and Evaluation
FQR	Formal Qualification Review
FQT	Formal Qualification Test
FSD	Full Scale Development
GFE	Government Furnished Equipment
ICS	Interpretive Computer Simulation
IOT&E	Initial Operational Test and Evaluation
I/O	Input/Output
JCL	Job Control Language
MAJCOM	Major Command
MIL	Military
OT&E	Operational Test and Evaluation
PCA	Physical Configuration Audit
PDL	Program Design Language
PDR	Preliminary Design Review
PMD	Program Management Directive
PMP	Program Management Plan
PQR	Preliminary Qualification Review
PQT	Preliminary Qualification Test
RFP	Request for Proposal
SAE	Software Acquisition Engineering
SAMSO	Space and Missile Systems Organization
SD	Space Division

## **ABBREVIATIONS AND ACRONYMS (Concluded)**

<b>SDR</b>	<b>System Design Review</b>
<b>SOW</b>	<b>Statement of Work</b>
<b>SRR</b>	<b>System Requirements Review</b>
<b>STD</b>	<b>Standard</b>
<b>QA</b>	<b>Quality Assurance</b>
<b>TDF</b>	<b>Test Development Folder</b>
<b>TEMP</b>	<b>Test and Evaluation Master Plan</b>
<b>TEOA</b>	<b>Test and Evaluation Objectives Annex</b>
<b>TER</b>	<b>Test Execution Record</b>
<b>TRR</b>	<b>Test Readiness Review</b>
<b>TVRM</b>	<b>Test Verification Requirements Matrix</b>
<b>T&amp;E</b>	<b>Test and Evaluation</b>
<b>UDF</b>	<b>Unit Development Folder</b>
<b>VDD</b>	<b>Version Description Document</b>
<b>V, V&amp;C</b>	<b>Verification, Validation and Certification</b>
<b>WBS</b>	<b>Work Breakdown Structure</b>

## 1. INTRODUCTION

This guidebook is written to assist Air Force Program Office engineering and management personnel in interpreting and applying the principles of Software Testing and Evaluation to the acquisition of airborne systems software. The guidebook concentrates on techniques and methodologies which have been successfully applied on software programs. It is general enough that it can be used on both large and small projects.

The guidebook describes the planning and testing which must be conducted to insure a successful software testing program. It presents guidelines and checklists to aid in reviewing the contractor's testing activities. The guidelines, checklists, and references supplement the information in government documents and summarize data from professional journals and books.

### 1.1 PURPOSE AND SCOPE OF SOFTWARE TESTING AND EVALUATION

The objective of software testing is to ensure that the software satisfies its specifications. The idea, of course, is that once all the errors are found and eliminated, the software satisfies its specifications. The elimination of all errors, except on simple programs, is unfortunately beyond the state of the art of software testing. This guidebook discusses techniques which have been successfully used to approach this goal for informal CPCI testing and for both Preliminary Qualification Test (PQT) and Formal Qualification Test (FQT).

Once errors are found, they must be eliminated. This process, sometimes called debugging, requires both analysis and detailed knowledge of the code developed. This topic is not addressed in this guidebook.



## **1.2 LIFE CYCLE RELATIONSHIP**

Software Testing and Evaluation is an activity which encompasses the entire life cycle. Planning begins when the software requirements are written; maintenance and retesting continues until the product is removed from the government's inventory. This guidebook discusses the testing activities from the initial planning through the retesting. Retesting is discussed in terms of its relationship to the system testing.

## **1.3 RELATIONSHIP TO OTHER GUIDEBOOKS**

A number of topics related to Software Evaluation and Testing are discussed in other guidebooks including

- **Software Requirements Analysis and Specification**  
Guidelines for the derivation and specification of requirements for software for an avionics system are presented here. The guidebook discusses the characteristics of software requirements which are needed for successful testing.
- **Verification, Validation and Certification**  
Independent Verification, Validation and Certification (V, V&C) is conducted by an organization other than the software contractor to provide assurance that computer programs will perform their mission requirements. The guidebook discusses V&V activities which can enhance software testing.
- **Quality Assurance**  
The Quality Assurance (QA) guidebook presents guidelines for monitoring software development activities to insure that the software program complies with the requirements of the contract, including software testing. The guidebook discusses monitoring activities needed to insure that the software has a number of desirable characteristics including testability. The characteristics of a successful QA program for software testing are presented.
- **Reviews and Audits**  
The information needed to help Air Force personnel plan, prepare, and conduct technical reviews and audits is presented in this guidebook. Guidance on the conduct of each review which may be required for successful software testing is given.

- **Configuration Management**

Successful Configuration Management (CM) is vital to software testing. The guidebook discusses CM activities needed for successful software testing.

- **Computer Program Documentation Requirements**

This guidebook provides guidance in the process of acquiring documentation of software development programs. Testing documents such as test plans, procedures, and reports are described.

#### **1.4 RELATIONSHIP TO VERIFICATION, VALIDATION AND CERTIFICATION**

V, V&C is conducted to provide the Air Force Program Office with systematic assurance that the computer programs will perform their mission requirements. Independent V, V&C is done by having an independent organization assess the adequacy of the software products. V, V&C may be performed by a test group which is part of the contractor's team, an independent contractor, or the government.

Verification is a technical review conducted during the time between the publication of the Part 1 CPCI Development Specification and completion of the Formal Qualification Test (FQT). Performed in parallel with the Software Testing and Evaluation, it involves independent reviews of the software requirements, software design, test plans, procedures, and results. Verification is conducted in parallel with Software Testing and Evaluation to complement, not compete with that activity. The verification activity may result in some modifications in the software testing that improve and enhance it.

Validation begins during the System Integration activities. It is concerned with reviewing and evaluating the integration and testing activities at the system level to provide assurance that the system satisfies the design criteria in the System or System Segment Specification and the Part 1 CPCI Development Specification.

Certification is an administrative procedure performed to insure that enough evidence is available to state with near certainty that the system will satisfy the user's needs. It is performed after the system testing has been completed. Certification embodies all the Software Testing and Evaluation, verification, validation, and Initial Operational Testing and Evaluation (IOT&E) activities that have been performed.

## **1.5 CONTENTS OF THIS GUIDEBOOK**

### **1.5.1 Section 1: Introduction**

Discusses the purpose and scope of the guidebook, its relationship to other guidebooks and the role of software testing and evaluation in the software life cycle.

### **1.5.2 Section 2: Relevant Documents**

Lists government documents relevant to software testing.

### **1.5.3 Section 3: General Guidance for Software Test and Evaluation**

Discusses the general characteristics of a software testing program including managerial considerations, planning and scheduling, test tools, and test control.

### **1.5.4 Section 4: Specific Guidance for Software Test and Evaluation**

Discusses the activities of Software Testing and Evaluation including informal unit and integration testing, formal PQT and FQT testing and retesting due to system testing. Presents guidelines for the reviewing and auditing of testing documents and results.

### **1.5.5 Appendix A**

Contains an annotated bibliography of books and articles on software testing.

## 2. RELEVANT DOCUMENTS

### 2.1 REGULATIONS, SPECIFICATIONS AND STANDARDS

- DIRECTIVES

DODD 5000.3	Test and Evaluation, 11 April 1978
DODD 5000.29	Management of Computer Resources in Major Defense Systems, 26 April 1976

- MILITARY STANDARDS

MIL-STD 483	Configuration Management Practices for System Equipment, Munitions and Computer Programs, 31 December 1970, Notice 1, 1 June 1971; Notice 2, 21 March 1979
-------------	--

MIL-STD 490	Specification Practices, Notice 1, 1 February 1969; Notice 2, 18 May 1972
-------------	---

MIL-STD 1521A	Technical Reviews and Audits for Systems, Equipments and Computer Programs, Notice 1, 29 September 1978
---------------	---

MIL-S-52779A	Software Quality Assurance Program Requirements, 5 April 1974
--------------	---

- AIR FORCE REGULATIONS

AFR 23-36	Organization and Mission-Field Air Force Test and Evaluation Center (AFTEC), 19 July 1976
-----------	---

AFR 80-14	Research and Development, Test and Evaluation, 16 July 1976
-----------	---

AFR 310-1	Management of Contractor Data, 30 July 1969, Change 1, 14 June 1971
-----------	---

AFR 800-14, Volume I	Acquisition Management, Management of Computer Resources in Systems, 15 September 1975, AFSC Supplement 1, 8 August 1977
----------------------	--

**AFR 800-14,  
Volume II**

**Acquisition Management, Acquisition  
and Support Procedures for Computer  
Resources in Systems, 26 September  
1975, AFLC Supplement 1, 18 October  
1976; Change 1, AFLC Supplement 1,  
31 March 1977**

• **SAMSO DOCUMENTS**

**SAMSO-STD 73-3**

**Standard Engineering Practices for  
Computer Software Design and  
Development, 6 October 1973**

**SAMSO  
Pamphlet 74-2**

**Quality and Reliability Assurance,  
Contractor Software Quality  
Assurance Evaluation Guide,  
1 September 1976**

### **3. GENERAL GUIDELINES FOR SOFTWARE TEST AND EVALUATION**

This section discusses the general activities that must be performed for successful software test and evaluation. The major considerations and trade-offs are described. Specific guidance for reviewing and auditing the contractor's testing program is presented in Section 4.

#### **3.1 PRELIMINARY CONCEPTS**

AFR 80-14 outlines policy and procedure for managing Test and Evaluation (T&E) activities during the development, production, and deployment of defense systems in the Air Force. It defines the two major categories of T&E activities as follows

- Development Test and Evaluation (DT&E) - all tests performed by the contractor from initial component tests to system integration and retest under the direction of the program office.
- Operation Test and Evaluation (OT&E) - performed primarily by, and under direction of, other AF agencies, i. e., the intended System Using Command (TAC, SAC, etc.), and/or Air Force Test and Evaluation Center (AFTEC).

For a weapon system procurement with embedded computers, software testing, as such, is normally limited to the DT&E phase. OT&E is system level oriented, "to provide a reasonable assessment of the system military utility in its intended operational environment". This guidebook material is primarily applicable therefore to DT&E type activities.

##### **3.1.1 Program Office**

The Program Office, headed by a Program Manager, manages the system (and its software) acquisition. It monitors contractor Software Testing and Evaluation and assists the contractor by

- Reviewing the planning of tests
- Participating in reviews and audits required by the contract
- Interfacing with the other major commands (particularly using and supporting commands)

On many programs the systems are acquired by contracting with a prime contractor who acquires the subsystems, often including software, through subcontracts. In this case, the Program Office manages the software acquisition indirectly through the prime contractor. The Program Office requires that the prime contractor conduct reviews and develop documents. The prime contractor may ask the subcontractor to participate in these activities, but the Program Office works with the subcontractor through the prime contractor.

The relationship between reviews and audits held periodically to monitor the software acquisition, and testing documentation available for review is shown in Table 3-1. Guidelines for reviewing the testing documents are presented in Section 4.

### **3.1.2 Contractor**

The software contractor has the responsibility to develop and test the software from the initial tests through the Formal Qualification Test (FQT) under the direction of either the prime contractor or the Program Office. The software contractor prepares the documents listed in Table 3-1, and schedules and conducts the reviews and audits as required on the program. In addition the contractor normally must provide for a Physical Configuration Audit (PCA), to establish the CPCI product baseline, and a Functional Configuration Audit (FCA), to determine the acceptability of the FQT.

If MIL-S-52779A is invoked on the contract, the prime contractor is responsible to establish a Software QA program. If the software is subcontracted, the prime contractor may direct the software subcontractor to establish it. The Software QA Program must identify all QA measures related to software testing. This includes reviews of software requirements, test plans, procedures and reports, monitoring of tests and certification of results, and identification of support hardware and software to be used. The Software QA Program also defines the procedure for reporting and correcting software deficiencies.

**Table 3-1. Software Related Testing Documents  
Reviewed at Reviews and Audits**

<b>Review or Audit</b>	<b>Software Documents Reviewed</b>
<b>System Design Review (SDR)</b>	<b>Preliminary CM Plan</b> <b>Preliminary QA Plan</b> <b>Preliminary Part 1 CPCI Development Specifications</b> <b>CPDP</b>
<b>Preliminary Design Review (PDR)</b>	<b>Final Part 1 CPCI Development Specifications</b> <b>Preliminary CPCI Test Plan</b>
<b>Critical Design Reviews (CDR)</b>	<b>Final CPCI Test Plan</b> <b>Preliminary CPCI Test Procedures</b>
<b>Test Readiness Review (TRR)</b>	<b>Informal review of the status of code, tools, facilities, personnel and CM procedures</b>
<b>Functional Configuration Audits (FCA)</b>	<b>CPCI Test Results</b>
<b>Physical Configuration Audits (PCA)</b>	<b>VDD, configuration index, change status reports, media, all above documents</b>



The prime contractor is also responsible for establishing a Configuration Management Organization (CMO) to control designated software products, and to establish a library of these products for all users. The CMO maintains logs of all changes to the baselined products, and initiates and controls procedures for changing them. If the software is subcontracted, the software subcontractor may establish a CMO to perform the above functions before the software is released to the prime contractor for system testing.

### 3.1.3 Testing in the Software Life Cycle

AFR 800-14 describes six phases of the software life cycle: analysis, design, code and checkout, test and integration, installation and operation support. As Figure 3-1 shows, the Software Testing and Evaluation activity begins early with the development of the software requirements, and then proceeds in parallel with the software development. Unit testing is shown as a software development activity, because it is generally conducted by the software developers. Retesting involves both the software developers and testers, because code must be both modified and retested. It is shown in the figure as a maintenance activity.

The reviews and audits shown in the figure are conducted as required. Except for the TRR, these reviews and audits may be required by invoking MIL-STD 1521A in the contract. The TRR is an informal review not required by MIL-STD 1521A. It may be required by including a requirement for it in the contract's Statement of Work (SOW).

The PCA is held for each CPCI to establish a product baseline. This may be held at the end of the FQT or the end of the System Integration Testing. Conducting the PCA at the latter time may allow changes to be made to the CPCI in a more cost effective manner.

A number of documents, related to Software Testing and Evaluation are prepared during the software's acquisition. They are listed in Table 3-2. Some administrative and managerial documents prepared



Table 3-2. Test Documentation

- **Test and Evaluation Objectives Annex (TEOA)**
  - **Purpose**

This document, an annex to the Program Management Directive (PMD), is responsive to the Decision Coordinating Paper (DCP), and states the specific Test and Evaluation (T&E) objectives that serve as the baseline for all evaluations of DT&E and OT&E programs.
  - **Originator**

The implementing command prepares the draft TEOA in coordination with the Air Force Test and Evaluation Center (AFTEC) for attachment to the PMD.
- **Program Management Plan (PMP)**
  - **Purpose**

The PMP is the implementing command's plan for the management of an entire system acquisition program. It is the directive for all participating commands.
  - **Originator**

The Program Office prepares the PMP.
- **Computer Resources Integrated Support Plan (CRISP)**
  - **Purpose**

This document identifies the computer resources required by the implementing, supporting, and using commands throughout the system life cycle and describes the plan for providing these resources. It is responsive to the direction in the PMD and PMP.
  - **Originator**

The CRWG, composed of representatives from the implementing, supporting, and using commands prepares this document.

**Table 3-2. Test Documentation (Continued)**

- **Test and Evaluation Master Plan (TEMP)**

- **Purpose**

- This is an overall test and evaluation plan designed to identify and integrate the effort and schedule of all T&E to be accomplished. The TEMP documents a coordinated position for all participants in the T&E program.

- **Originator**

- The Program Office (PO) prepares this document.

- **Computer Program Development Plan (CPDP)**

- The CPDP is developed in accordance with DI-S-30567A.

- **Purpose**

- This plan identifies the actions needed to develop and deliver computer program configuration items and necessary support facilities. This document is discussed in detail in paragraph 3.2.

- **Originator**

- The contractor normally prepares this document.

- **Part 1 CPCI Development Specifications**

- These documents (also called TYPE B5) are developed in accordance with MIL-STD 483.

- **Purpose**

- These specifications contain the requirements to which the software contractor is contractually obligated to develop the software.

- **Originator**

- The contractor prepares these specifications.

**Table 3-2. Test Documentation (Continued)**

- **Software Standards and Procedures**

- **Purpose**

These documents state the program's software standards including documentation, design description, programming, and testing standards.

- **Originator**

This is a collection of documents that may be prepared by the contractor, the Air Force, or a combination of both. These may be contractual or simply imposed as a guide.

- **Test Requirements Verification Matrix**

- **Purpose**

This matrix identifies the software component(s), CPC and/or CPCI, which implements each software requirement. DI-S-30567A requires that the CPDP define the technique used to assure traceability to requirements. This document provides a way to comply with that requirement.

- **Originator**

This is an internal document that the contractor should prepare in conjunction with completion of the B5 specification, Section 4 (Note: MIL-STD 483, Notice 2, also requires a matrix).

- **Test Plans**

CPCI Test Plans are written in accordance with DI-T-3703A or DI-T-30715. Test plans for informal (e.g., unit development/integration) testing are internal contractor documents.

- **Purpose**

The formal documents provide detailed, coordinated, integrated, and time-phased planning for software T&E. A separate test plan is written for each individual testing activity defined in the SOW.

- **Originator**

The contractor prepares these plans. The Program Office reviews and approves the CPCI plans for the PQT's and the FQT.

**Table 3-2. Test Documentation (Concluded)**

- **Test Procedures**

CPCI Test Procedures are written in accordance with DI-T-3703A or DI-T-30716. Test procedures for informal testing are internal contractor documents.

- **Purpose**

The formal documents, prepared for each CPCI, present the detailed steps needed to conduct each test specified in the test plan.

- **Originator**

The contractor prepares the procedures. The Program Office reviews each CPCI procedure for the PQT's and the FQT.

- **Test Reports**

CPCI Test Reports are developed in accordance with DI-T-3717A. Test reports for informal testing are internal contractor documents.

- **Purpose**

Test reports are written to document the analysis performed and results obtained from each test. They provide a record of the software's performance to show that the software satisfies its specifications.

- **Originator**

The contractor prepares these reports. Reports of each PQT and the FQT are reviewed and approved by the Program Office.

- **Version Description Document (VDD)**

This document is prepared in accordance with DI-E-3421.

- **Purpose**

The VDD is prepared for each release of version or interim change of a CPCI. It identifies the item delivered and records all changes made to it.

- **Originator**

The releasing contractor prepares each VDD.

by the Air Force are discussed to show the flow of software testing directives from the Program Management Directive to the individual tests.

The Computer Program Development Plan (CPDP) is required for all software acquisition programs. Other documents prepared by the contractor may be required by referencing the appropriate Data Item Description (DID) in the Contract Data Requirements List (CDRL). Contractor internal documents may be obtained as identified on the Data Accession List.

### **3.2 PLANNING CONSIDERATIONS**

A successful software testing program requires early and careful planning, an activity which continues through the entire acquisition activity. Test plans, written at progressively lower levels, provide a mechanism to focus the planning concepts and to provide visibility to the contractor and the program office. One of the first of these test plans is actually part of the Computer Program Development Plan (CPDP), a managerial document, which is discussed below. Other planning documents, more closely related to individual testing activities, are discussed in paragraph 4.

#### **3.2.1 Computer Program Development Plan**

The Computer Program Development Plan, required by AFR 800-14 for all software acquisition programs, is an overall plan to identify the actions needed to develop and deliver the computer program configuration items and necessary support resources. The document is usually written during the analysis phase of the system development or as a response to an FSD RFP. According to AFR 800-14, the items affecting the testing that should be addressed are

- The organization, responsibilities, and structure of the group(s) that will be testing the computer programs.
- The resources required to support the test of computer programs.
- The general procedure for reporting, monitoring, and resolving computer program errors and deficiencies during testing (if not part of the QA plan).

- The methods and procedures for collecting, analyzing, monitoring, and reporting on the timing of time critical computer programs.
- The methodology for insuring satisfactory testing (including informal testing).

Specifically, the CPDP should contain the items listed in Table 3-3. These items, found in DI-S-30567A, are detailed in the following paragraphs.

### 3.3 TEST CONDUCT AND CONTROL

#### 3.3.1 Test Design

The test design, which forms the testing philosophy discussed in the CPDP, is usually based on two techniques of software development and testing, bottom up and top down. Since the developmental approach used significantly impacts the testing philosophy, the developmental techniques and their relationships to testing are discussed.

The bottom up development starts with the design and coding of individual routines. Usually the lowest level routines considered most important to the application are developed first; the higher level calling routines are then coded. Generally the interfaces between the routines will not be completely defined when the lower level routines are developed so modifications to lower level routines will be required as the higher level routines are developed. This iterative process of developing higher level routines and modifying lower level ones as needed continues until all the software is developed.

Bottom up testing begins by testing each individual routine as it becomes available. Routines are then combined and tested in large units of code, continuing until the entire software product has been tested. Bottom up testing requires that dummy routines called drivers be developed to simulate the routines which call the modules being tested.



**Table 3-3. Items in the CPDP Related to Software Testing**

- Definition of the integration and testing philosophy, how it leads to PQT and FQT.
- Identification of the testing activities and their relationship to the Work Breakdown Structure (WBS).
- Definition of the testing organization, its responsibilities and relationship to the other contractor's organizations.
- Identification of standards for development and testing, including mechanisms for verifying compliance with the standards.
- Identification of how traceability between requirements and software is achieved.
- Resources needed including special hardware, software, GFE and data.
- A listing of the testing documents to be produced including when they will be available.
- Identification of test tools needed and how they are validated.
- A discussion of the control mechanisms needed for software testing including method of documenting software errors and special aspects of Configuration Management not addressed in the Configuration Management Plan.

**Note:** The CPDP is the only source of information on unit level testing.

The top down development begins with the definition of the functions of the system. The logic controlling the sequencing of the functions and their interfaces is then defined. The top level control routines are then defined and developed. The above process is then repeated for the next lower level of code, continuing until all the routines have been developed and integrated.

The top down approach forces interfaces to be defined early, and should reduce the changes needed to existing routines. Unfortunately, interfaces are rarely completely defined at each level, so modifications are generally made to the higher level routines as the lower level ones are developed. Top down testing begins by integrating and testing the highest level routines, using dummy routines called stubs, to simulate the missing called routines. After these routines are integrated, the process is repeated for routines at the next lower level. Again stubs simulate the called routines. This process continues until all the software is successfully integrated.

The nature of development testing will change significantly in a top down approach. Development test planning will have to treat checkout and integration as a single process. It will no longer be possible (unless exception is taken to doing everything top down) to checkout the internal working of a routine before making sure that it fits into the system. Therefore, the testing design, which forms the testing philosophy, should be stated in the CPDP. Most testing programs use a combination of the top down and bottom up approaches. High level routines are developed and tested in a top down manner. A few selected routines are coded and developed from the bottom up, especially critical routines and I/O routines. The advantages and disadvantages of each approach are summarized in Table 3-4.

### **3.3.2 Control of Software Tests**

Initial development testing control is quite informal. The documentation prepared for tests executed by the programmers is generally

**Table 3-4. Advantages and Disadvantages of Bottom Up and Top Down Development and Testing**

Bottom Up	
● Advantages	● Disadvantages
<ul style="list-style-type: none"> <li>● High risk routines tested early</li> <li>● Utility routines are developed early and tend to be common to the program</li> <li>● Easier to control testing conditions</li> <li>● Development of top level structure can be delayed, allowing selection of a specific machine to be made later</li> <li>● Drivers simple to develop</li> </ul>	<ul style="list-style-type: none"> <li>● Testing difficult because interface problems and system requirements not addressed early</li> <li>● Hard to maintain visibility of entire software system</li> <li>● Difficult to change because interfaces may be "kludged"</li> <li>● Data base structure not addressed early can result in "kludged" data base</li> <li>● System cannot be executed until late in testing</li> </ul>
Top Down	
● Advantages	● Disadvantages
<ul style="list-style-type: none"> <li>● System executes early in development and testing</li> <li>● Data structure and interface problems addressed and resolved early</li> <li>● Testing can be done in parallel at more than one level</li> <li>● Easier to maintain visibility of the entire software system</li> </ul>	<ul style="list-style-type: none"> <li>● Low level high risk modules not developed early</li> <li>● Multiple utility routines may exist or redesign may be needed to use common utility routines</li> <li>● Developing top level structure early may force project to specific machine(s) too early</li> <li>● Stubs must be developed - they may be complicated</li> <li>● Testing conditions may be hard to control</li> <li>● Coding of top level routines may begin before the design is completed</li> <li>● Error conditions may be more difficult (costly) to exhaustively exercise</li> </ul>

maintained and controlled by the individuals performing the testing. While the test plans and procedures are usually reviewed by the contractor, changes generally do not require approval by anyone outside the programming team.

An increased level of control is usually placed on the informal testing when an independent test team becomes involved. Plans and procedures are published for internal contractor review, and are made known to the Program Office via the Data Accession List. This documentation makes the following visible to the contractor

- extent of testing
- time phased resource needs
- schedule

It also provides a mechanism to identify problems early. If the milestones on the published schedule are not met, this is known, and corrective action can be taken. Documentation is discussed in the Guidebook on Computer Program Documentation Requirements.

Changes to these test plans and procedures may be controlled by the contractor at his discretion. Some of these test plans and procedures may be controlled by the contractor's CMO, whose activities are discussed in the Guidebook on Configuration Management.

Test plans are written by the contractor for formal testing, PQT and FQT, and submitted to the Program Office for approval prior to the testing. After the test plan is approved, changes should require approval by the Program Office. A separate test procedure is written for each qualification test and submitted to the Program Office for review prior to the testing. The test procedures however may be updated by the contractor as the testing proceeds to reflect the actual testing performed. Formal test results are documented by the contractor for review by the Program Office. Each of these testing documents should be identified in the CPDP.

### **3.3.3 Control of the Software Products**

Software being developed and tested by the originating programming team is informally controlled, since the product is not available except to team members. Changes are made at their discretion and internally coordinated.

When software products are released for use by organizations other than the teams that developed them, the products must be controlled by a central organization. This organization called Configuration Management Office (CMO) will establish a library, usually automated, with appropriate access and update controls. Details are discussed in the Guidebook for Configuration Management. Some typical items usually contained in the CM library are shown in Table 3-5.

The controlled software products are placed in a library for use by the software testers. The library version, however, can only be modified under controlled conditions (e.g., approval by the CMO). After a software product has been placed under library control any requests for changes must be documented and submitted to a change board. The contractor's internal engineering change board or Configuration Control Board (CCB) performs this function. If a subcontractor develops the software, there may be two internal CCB's, the subcontractor's CCB and the prime contractor's CCB, both of which must concur with any change.

Approved changes are given to the software developers for implementation. The software product is then released for control by the CMO.

To change a controlled software product, a Discrepancy Report (DR) (or contractor change request form) is completed and sent to the CCB. This form should contain the information listed in Table 3-6.

**Table 3-5. Candidates for Control by  
Configuration Management**

- **Contractually deliverable code - any coded and debugged portions of a deliverable program**
- **Data base dictionaries**
- **Data base values**
- **Directives (e.g., JCL statements) to assemble or compile code**
- **Directives to create load images**
- **Directives to execute code**
- **Directives to execute tests (test scripts), especially those for PQT and FQT**
- **Test tools listed in paragraph 3.4, especially those used in PQT and FQT**
- **Patches (or quick fixes) to a controlled software product**
- **Compilers and assemblers (or modifications to compilers and assemblers) needed to create object code for deliverable software**
- **Test results (successful and unsuccessful)**

**Table 3-6. Information of Discrepancy Reports**

- **Description of the Problem**
  - Identifier (Discrepancy Report Number)
  - Originator and date
  - Affected software product
  - Type of problem
  - How detected
  - Testing level and test (if any) at which problem was detected
  - Hardware and software configurations used
  - Reason for change
  - Test(s) affected, if any
  - Proposed change, if available
  - Affected document(s) (i. e. , specification(s), if any)
  - Internal CCB action (accept and assigned to ---, defer until ---, reject)
  - Signature of authorized member of the internal CCB
- **Disposition of the Problem**
  - Responsible organization
  - Actual change made
  - Software product(s) changed
  - Approval signatures and dates
- **Library Change**
  - Library(ies) changed and date, new version number(s)
  - Retesting done
  - Approval signatures and dates (by Retesting Test Team, QA, and CMO)

Each DR must be resolved. To insure this, a Discrepancy Report Log (DRL) summarizing the status of each DR should be maintained by the internal CCB. The DRL should contain the information listed in Table 3-7.

**Table 3-7. Information on Discrepancy Report Log**

- |  |
|--|
| <ul style="list-style-type: none"><li>● DR Identifier and Date</li><li>● CCB Action</li><li>● Status (open, closed on -----)</li></ul> <p>If DR is accepted</p> <ul style="list-style-type: none"><li>● Library(ies) to be changed, New version number(s)</li><li>● Expected or actual date of change</li><li>● Responsible organization</li></ul> |
|--|

The DR forms may be used to gather statistics on software errors. The number of errors, their type, and resolution by product and as a function of time, can be obtained. This can be used in estimating the quality of the software products and in isolating potential trouble spots. A product with an inordinate number of errors may require special attention, more testing or possibly a redesign. Summaries of these data may be made available to the Program Office via the Data Accession List.

Temporary patches to controlled software products may be required to continue the testing. These patches may be needed to

- repair software errors
- add overlooked but necessary capabilities
- log data needed for analysis
- bypass hardware problems
- modify the software configuration to execute certain tests



Since the patches can affect the testing results they should be used carefully. Whenever possible the software patch should match the eventual permanent software change (if there will be one).

These patches should be documented and controlled items. The following information should be recorded for each patch

- Identifier
- Reason for Patch
- Originator of Patch and Date
- Software Product(s)/Configuration(s) Affected
- Associated DR, if any
- Resolution of Patch
- Listing of Patch

Testing runs should include a list of patches used, possibly using a test tool to generate the list automatically.

The method by which software errors are detected, documented, and corrected is stated in the CPDP. The method of controlling the software products is stated in the overall Configuration Management Plan or in the CPDP. The CPDP includes special aspects of software configuration management not in the overall CM plan.

### 3.4 TEST TOOLS

Many tools have been developed to aid in the testing and evaluation of software. The tools required should be identified early in the program so appropriate resources can be allocated to their development. DI-S-30567A requires the contractor to identify the tools in the CPDP, stating their purpose, application, and validity.

The application of the tools should be carefully stated since they can affect the software's performance. They can

- introduce errors into the results
- increase the memory required by the software
- alter the timing of the code

The procedure for qualifying tools and their use during software development must be stated. Tools may be qualified by

- successfully completing a testing program.
- establishing that they have been used successfully on other software development programs.

MIL-S-52779A may be invoked to require the contractor to describe how existing test tools were tested and qualified and how proposed test tools will be tested and qualified prior to their use (SAMSO P 74-2).

Simple tools or those applicable only to one testing team are generally developed by that team. More complicated tools or those useful to many test teams (simulations, for example) are usually developed by a test tool development team. Advantages and disadvantages of each approach are presented in Table 3-8.

Test tools may be used for the following software testing activities

- enforcement of software standards
- requirements verification
- product management
- error detection and performance analysis
- production of test scripts

Code or test auditors can be executed during software development or testing to insure that the code satisfies certain standards. The successful execution of these programs can be made part of the criteria for the release of code or completion of testing.

Test Requirements Verification Matrices provide traceability from requirements to test cases, and substantiate that each requirement has been allocated to a test. The matrices should contain the following information

- Date
- Specification (and Revision Number, if any)

**Table 3-8. Advantages and Disadvantages of Development of Test Tools by Testers or Another Organization**

Test Tool Development	Advantages	Disadvantages
<ul style="list-style-type: none"> <li>by Test Team or users</li> <li>by Test Tool Development Group</li> </ul>	<ul style="list-style-type: none"> <li>Tool tailored to needs of user</li> <li>Reduce redundancy</li> <li>Tools may be general enough to apply to other projects</li> <li>More visibility - forces plans to allocate resources to design, development, and testing</li> </ul>	<ul style="list-style-type: none"> <li>Tool too narrowly designed to be used by others</li> <li>Same tools developed independently by other groups</li> <li>Tools not developed by specialists</li> <li>Interface with users requires plans, procedures - may be costly</li> <li>Tools may not exactly meet needs of users</li> </ul>

- Paragraph Number
- Test case(s)
- Testing level
- Method of Verification
- Software Component (CPC/CPCI)

Top down testing uses stubs to replace called routines. The stubs may merely return control to the called routines or may include the capability to

- print messages indicating it was called
- return variables (typical or computed values)
- approximate the size of the code it replaces
- approximate the timing of the code it replaces

These more complex or smart stubs can provide a better approximation of how the entire software product will perform. They require more resources to develop, however, and must be tested. The determination of how smart each stub should be depends upon the complexity and the importance of an accurate representation of the missing code.

Product Management, while not a part of software testing, is vital for success. Therefore, tools used for this activity are listed here. In particular the management of the software patches generated by the testing activity requires close cooperation between the testers and product managers, most likely configuration management.

Tools for error and fault isolation are clearly required. Selected dumps and traces, appropriately formatted, and data logging tools will be the most important software tools for much of the early integration testing.

The development of tools for performance analysis may use most of the resources for test tool development. In particular, the development of a simulation may be a major effort. It may itself be deliverable software. An existing simulation may be used with an interface developed for the new software. For example, a software interface may be built for an existing missile simulation to evaluate new onboard guidance software.

Summaries using statistics, graphs, and plots may be important to evaluate programs with large amounts of output. In some cases, data reduction languages may be developed to aid in writing such test tools.

As the testing continues, the production of test scripts will become important. In retesting or regression testing, test scripts which automatically or at user's selection execute a number of test cases and summarize the results, are very useful.

Examples of each type of test tool are listed in Table 3-9.

### 3.5 MANAGEMENT CONSIDERATIONS

#### 3.5.1 Organization

Any contractor should generally have a staff organization and a product line organization headed by a quality assurance (QA) manager, and the program manager respectively. There may be a third organization, the contractor's Configuration Management Office (CMO), or the CM and the QA function may be combined.

For software development the QA manager has the responsibility to provide an independent assessment that the software complies with the requirements of the contract. To provide this independent assessment, the QA organization must be independent of the organization charged with developing the software. The QA organization reviews both the development process and the products produced for compliance with the standards and procedures established for the program. The role of the QA organization during software development is discussed in the Guidebook for Quality Assurance.

**Table 3-9. Test Tools**

Purpose	Tool	Description
<b>Enforcement of Software Standards</b>	<b>Code Auditor</b>	Checks that appropriate structure and coding constraints were observed, (structured programming, adequate comments, few GOTO's, etc.)
	<b>Test Auditor</b>	Determines number of branches exercised and not exercised during a series of test runs.
<b>Requirements Verification</b>	<b>Test Requirements Verification Matrix</b>	Lists each requirement and the test(s) which demonstrate that the software satisfies the requirements.
<b>Product Management</b>	<b>Checksum or Comparator</b>	Compares two sets of code or data to determine that they are identical.
	<b>Software Patch Manager</b>	Creates files of software patches for use by test teams.
	<b>Software Library Manager</b>	Aids in creating and maintaining files of baselined software products for users.
<b>Error Detection and Performance Analysis</b>	<b>Editor or Static Analyser</b>	Analyses source code for coding errors and obtains information used to check relationships between sections of code, determines data usage (elements input, computed, used, output), checks error prone constructions, determines inaccessible instructions.
	<b>Flow Chart Generator</b>	Creates a flow chart from the code. It may show where data elements are set and used.
	<b>Instrumenters</b>	Inserts instructions into code to record the value of data elements during execution.
	<b>Dump and/or Data Logging</b>	Records the value of data elements under specific conditions. The data may be formatted and may be selected by the tester.
	<b>Pathfinders</b>	Records the history of routines called prior to the occurrence of a specified condition (usually an error condition).
	<b>Simulator (or Simulator Interface)</b>	Simulates external devices or environments (or provides an interface between an existing simulator and the software) to provide a more realistic environment for testing.
	<b>Data Reduction Programs</b>	Provides a language that allows easier access to output data.
	<b>Summaries, Plots, Graphics, Statistical Computations</b>	Summarizes software performance for analysis, presentation, and generation of reports.

**Table 3-9. Test Tools (Concluded)**

Purpose	Tool	Description
<b>Error Detection and Performance Analysis (cont)</b>	<b>Dynamic Analyzer</b>	Collects statistics on software characteristics, such as paths executed, frequency of execution of sections of code (routines, CPC's, CPCI's), percentage of total execution time used by sections of code, queue lengths, and waiting times.
	<b>Timing Analyzer</b>	Monitors and reports execution time of program elements.
	<b>Driver</b>	Simulates calling routines to allow called routines to be executed.
	<b>Stubs</b>	Simulates called routines not yet developed to allow developed routines to be tested.
	<b>Computer Simulator or Interpretative Computer Simulator (ICS)</b>	Simulates the execution characteristics of a target computer (for which the software is written). The program is executed on another computer (a host computer).
	<b>Emulator</b>	A computer firmware tailored to operate exactly like the computer it replaces.
	<b>Data Analyzer</b>	A program that checks the definition and use of data items for consistency, proper specification and use.
	<b>Hardware Monitor</b>	A hardware device that obtains signals from probes inserted in a host computer's circuitry.
	<b>Interface Checker</b>	A computer program that checks the ranges of variables to ensure compliance with Interface Design Specifications.
<b>Production of Test Scripts</b>	<b>Test Data Generators</b>	Creates test data, usually in a form directly accessible to the software being tested.
	<b>Job Control Language (JCL) Data</b>	A sequence of JCL allowing a tester to execute one or more tests and obtain output easily (especially useful for retesting, when the same tests are executed repeatedly).

On large software projects, the software contractor's program manager may establish teams within his organization to assess the quality of the software products or to control some products during their development. This can provide him visibility which he may otherwise not have. This does not obviate the need for separate CM and QA organizations, however.

The contractor should also have separate organizations for software development and testing, because

- It is difficult for developers to find errors in their own products.
- It is difficult to review and critique one's own work.
- Independent testers may have a different interpretation of ambiguous requirements. Discovering and resolving this early is cost effective.

The testing may be conducted by the software contractor's QA organization or by the software contractor's program organization. If the second approach is used, the program manager directs two independent organizations, one charged with software development and the other with software testing and evaluation. Another valid approach is for the program manager to direct the unit testing and CPCI integration, while the QA organization performs the system testing and conducts the retesting.

Software testing is done by a number of individual test teams or individuals, each with the responsibility for a particular activity. The software development organization is responsible for the routine or unit level testing. The software developers may perform some integration testing. For example, they may develop and integrate an environmental simulator, used by the software testing organization to drive operational software. In this case, test teams separate from the programming teams may be established within the software development organization.

Generally, the independent testing organization is responsible for additional testing performed using significantly larger amounts of code. These testing activities, forming the testing levels, may include

- unit testing
- CPC testing



- CPC integration
- integrated CPCI testing

The CPC and CPCI integration activities may be one testing level, or they may each be comprised of multiple testing levels. The number of testing levels, which should be identified in the CPDP, depend on

- resources available
- size of software
- complexity of interfaces
- innovativeness and difficulty of the software
- criticality of the software

A small program may require only two levels of testing, unit testing and one set of integration tests, while a complex program may need more levels of integration testing each using significantly more complicated portions of the product.

At the individual manning level, some software developers may be included on the testing organization's test teams. This has the following advantages

- gives the development organization some responsibility for successful testing
- provides the testing organization with expertise on the details of the products
- provides personal contacts with the development organization

To maintain the independence of the testers, these people should not manage the testing activities.

The testing responsibilities of the development and testing organizations should be clearly stated in the CPDP. Software performance criteria which must be satisfied to release software to the testing organization should be defined and verified when the products are released.

### **3.5.2 Resources and Facilities**

The CPDP should identify time phased needs for resources and facilities for software testing and evaluation, including both contractor furnished equipment (CFE) and government furnished equipment (GFE). Periods of heavy demands on equipment, especially heavy demands on computer facilities, should be anticipated. This demand may be due to activities other than software test execution. These activities may include

- software patch generation
- simulation development or simulation interface development
- test plan, procedure, and data analysis report generation
- test tool development or conversion
- dry runs
- preparation of test scripts including data bases for testing

These activities must be considered in determining computer resources required for software testing for they may require considerable resources.

### **3.5.3 Schedule**

Three approaches have been used to monitor software testing.

- assessing the status of each test
- determining the number of completed tests
- assessing the rate of occurrence of errors

The status of each test may be independently assessed to monitor the progress of the testing. This is satisfactory if care is used. Often a test may appear to be nearly complete, when in fact it is not. As a result, a testing activity may be reported as almost completed when it isn't.

A testing activity may consist of executing a given number of tests. The ratio of the number of completed tests to the total number may be used to measure the progress of the work. A problem with this approach is that initial tests may be completed quickly. This may lead one to believe that the testing is nearly complete, while in fact the few remaining tests may require considerable resources to complete. If this monitoring approach is used, the status of each test must be carefully assessed.

During a testing activity, most errors will be found early. The rate of occurrence will rise to a maximum and then decrease. This may be used to monitor the progress of the testing. In fact, when the rate of occurrence of errors drops below a predetermined number, the testing activity may be declared complete. A point of diminishing returns has been reached and, according to this theory, resources can then be better used for other activities. This approach has two problems. The error rate which signals the end of the testing must be predetermined, and when this occurs tests may not be executed properly. While continued testing may not uncover many more errors, stopping without successfully executing each test is not very satisfying, and should not be allowed for PQT's or the FQT. This method of monitoring progress must be used with care.

#### **4. SPECIFIC GUIDANCE FOR SOFTWARE TEST AND EVALUATION**

##### **4.1 TEST ENGINEERING AND SPECIFICATION**

The quality of the CPDP and Part 1 CPCI Development Specification<sup>†</sup> are important factors in the success of the testing effort. Therefore, they must be carefully reviewed.

The CPDP, discussed in paragraph 3.2.1, may be reviewed at the SDR to insure that the contractor plans to conduct a responsive test program to reduce the technical risks of the software development. Since the CPDP defines the contractor's approach to software development for the program, the entire document should be thoroughly reviewed. A checklist is given in Table 4-1 to aid in reviewing the parts of the CPDP which pertain to Software Testing and Evaluation. These parts may be scattered through the CPDP, so some paging may be necessary to find this information. The reviewer should be able to answer each question in the checklist. A negative response to any question in the checklist should be resolved with the contractor to the reviewer's satisfaction before the document is accepted. If the CPDP is a proposal tool, Table 4-1 can be used to aid in source selection.

A Preliminary Part I Development Specification for each CPCI should also be available at the SDR. A review checklist appears in Table 4-2. Questions which cannot be satisfactorily answered should be resolved with the contractor before the review is completed. It has been demonstrated repeatedly that inadequate software requirements or specifications cause serious problems, so it is critical that the specification be clear, complete, unambiguous, and testable.

---

<sup>†</sup>Part 1 CPCI Development Specification includes Interface Control Specifications.

**Table 4-1. CPDP Review Checklist, Testing Sections**

- **Testing Structure**
  - Are the testing levels (including informal tests) defined? Are the scope and objectives for each testing level stated?
  - Are the test support software and hardware needed for testing identified?
  - Are the organizations responsible for each testing level identified? Are their responsibilities clearly defined? Are the testers independent from the software developers?
  - Are independent internal reviews planned? Are the organizations responsible for these reviews identified? Will these reviews include
    - software design
    - test plans
    - test procedures
    - test results
  - Are standards given for these reviews? Are correction procedures stated?
  - Is a schedule shown for the testing activities?
  - Are formal reviews and audits identified? Is a schedule shown for each? Is the documentation to be available for each review and audit listed?
  - Is an independent Quality Assurance organization identified? Do you know what QA's role in software testing will be from reading the CPDP or the Software QA Plan, if MIL-S-52779A is invoked in the contract? Do you know what its relationship to CM is?

**Table 4-1. CPDP Review Checklist, Testing Sections (Continued)**

- **Testing Control**
  - Is an organization identified to control software products and documents?
  - Are the products and documents that will be controlled identified?
  - Does the CPDP state when these items will be baselined?
  - Does the CPDP state how items will be controlled prior to formal baselining?
  - Is a mechanism given to implement this control stating
    - how changes to controlled items are proposed
    - how requests for changes are reviewed and resolved
    - how controlled items are changed
    - who has the responsibility for each of the above activities
- **Testing Standards**
  - Are standards listed for the testing activities?
  - Is an organization identified with the responsibility to enforce the standards? Does this organization "sign-off" on the completed test procedures and report?
  - Is a methodology given to enforce the standards? (The methodology may be a set of guidelines to be published in another document. If this is the case, that document should be identified).
  - Is a mechanism identified to map CPCI Development Specification Requirements to test cases?

**Table 4-1. CPDP Review Checklist,  
Testing Sections (Concluded)**

- **Testing Objectives and Priorities**
  - **Is the testing philosophy defined? Is it shown how this leads to PQT's and FQT?**
  - **Is the importance of testability relative to other software characteristics discussed?**
- **Documentation**
  - **Is the documentation approach discussed?**
  - **Is each contractually deliverable testing document identified?**
  - **Is the organization responsible for each contractually deliverable document shown?**

**Table 4-2. Part 1 CPCI Development Specification Review Checklist**

- **Is each requirement consistent? Can you find two or more requirements which, under some conditions, force the software to do something logically impossible?**
- **Is each requirement clear? Do you understand exactly what it means?**
- **Is each requirement unambiguous? Can you think of more than one interpretation for the requirement?**
- **Is each requirement testable? Requirements should state range of acceptable inputs, processing to be performed, and output. Can you think of a test which could be used to show that the software verifies the requirement? If the test requires a computer run, you must be able to specify input, output, and acceptance criteria.**
- **Are the requirements complete? Can you completely map each requirement in the system specification, which is allocated to software, to one or more requirements in a CPCI Development Specification? Can you think of a software requirement in the system specification which is not allocated to a requirement in the CPCI Development Specification? Could an input outside the range of acceptable values occur? If so, is the software's response specified?**
- **Are all the requirements necessary? Can you find a requirement in the CPCI Development Specification which cannot be referenced to one or more requirements in the system specification?**
- **Is each requirement feasible? Should the software perform the function stated? Can the requirement be satisfied in the intended system?**
- **Is each requirement allocated to a test? Is there a matrix in the CPCI Development Specification which identifies the following for each requirement**
  - **method of verification (inspection, analysis, simulation, operational test)**
  - **testing level (unit, software integration, system integration)**
- **Are these mappings correct? Can you think of a situation in which the requirement cannot be verified by the method shown or at the testing level listed?**
- **Are any timing or sizing requirements near the limits of the system? If so, this should cause alarm. The software will be hard to modify and expensive and risky to develop.**



## **4.2 SOFTWARE TESTING AND EVALUATION**

### **4.2.1 Unit Tests**

Unit testing, performed by the software development organization on each routine, is the lowest level of software testing. It is designed to make each executable section of code, module or routine, as error free as possible and is oriented to finding errors which commonly occur in the development of code, such as

- coding errors
- computational errors and inaccuracies within a routine
- inadequate input data specification
- incorrect output formatting and content
- logical errors within the unit of code

This testing is also used to verify requirements which can be allocated to the unit level.

The software development organization is responsible for this testing activity. That organization is responsible for all the documentation of the testing for that routine, including generating the test plan, procedure, input and expected output data, analyzing the output, and documenting the results.

The software contractor should have established testing standards for this activity before it begins. These standards may, for example, include the following:

- Each instruction is executed at least once.
- Each branch of each decision statement is executed at least once.
- All formats and options are exercised. Lower and upper limits are tested. Out-of-range values are tested and error messages are formed where appropriate.

- All error messages are exercised.
- All computational logic is manually checked.
- Table limits are exercised, including cyclic tables through their cycle points.
- All data input options are exercised.

These standards require two things. The person defining the test cases must know the structure of the routine being tested, and must be able to control its execution through input.

Test cases at this level are usually defined by a detailed analysis of the code being tested. The test cases typically insure that the code executes as it was designed. They do not generally consider whether the routine was designed correctly. To avoid this problem, an independent group, the testing organization or QA, may review the design of the CPCI, and prepare a list of functional capabilities which the routine should satisfy. The unit testing, of course, should show that the routine does have the listed capabilities.

As with any testing activity, test plans, procedures, and reports should be prepared. In this case the documentation is usually informal; it is available for review, but is often not published. The test plan may be a single document covering all the unit tests. The test procedures prepared for each test may merely list the input and expected output. The report may consist of the output with certification by the contractor's QA organization that it matches the expected output. The testing documentation may be kept with the developmental documentation for the routine. On some programs, this information is maintained in a Unit Development Folder (UDF) for each group of related routines. This provides a controlling mechanism for the code at this stage of development. Figure 4-1 is an example of a cover sheet for a UDF.

PROGRAM NAME \_\_\_\_\_

UNIT NAME \_\_\_\_\_ CUSTODIAN \_\_\_\_\_

ROUTINES INCLUDED \_\_\_\_\_

SECTION NO.	DESCRIPTION	DUE DATE	DATE COMPLETED	ORIGINATOR	REVIEWER/ DATE
1	REQUIREMENTS				
2	DESIGN DESCRIPTION PRELIM: "CODE TO"				
3	FUNCTIONAL CAPABILITIES LIST				
4	UNIT CODE				
5	UNIT TEST PLAN				
6	TEST CASE RESULTS				
7	PROBLEM REPORTS				
8	NOTES				
9	REVIEWER'S COMMENTS				

Figure 4-1. Sample UDF Cover Sheet

Much of this testing may use automated test tools described in paragraph 3.4. A testing standard may be defined for the program, requiring that each unit test use a single test driver. The execution of certain test tools, such as code auditors or dynamic analyzers, may be required for each test. This can be used to enforce standards such as adequate comments, structured programming, and the execution of each branch of code.

Automated test tools can simplify the documentation needed. If the same tools and supporting facilities are used for each unit test or group of tests, that information need not be repeated for each routine. A description of each test, its input, and expected output should be included, however.

During the unit testing, the tester will generally be the only person using the code. Therefore, the code will generally not be controlled. If these tests show that the code satisfies requirements, it must be shown that the code executed is the same as the software product which is put in the contractor's program library. Checksum programs can be used to do this.

Before testing begins, reviews may be conducted by the contractor; these include<sup>†</sup>

- Desk checking - an inspection of the code by the programmer.
- Code inspection - a review of the design and code of a routine by a team of programmers.
- Walkthrough - a detailed review of the design, coding, and testing documentation by examining the execution expected of a set of input data.
- Rating by colleagues - a review of the routine by other programmers who rate it and make suggestions for improvements.

---

<sup>†</sup> A number of publications provide checklists for these reviews. Appendix A lists some of these documents.

The programmer should conduct the desk checking in the course of coding the routine. The other reviews may be conducted by the contractor for some or all of the code developed. Since conducting the latter three reviews for all routines may be expensive, the contractor may elect to conduct one or more reviews for selected routines (critical and/or chosen at random).

The Program Office may not have the resources to participate in these reviews. In fact, participation by the Program Office may hinder a thorough review at this low level. The designers may be reluctant to discuss problems with the Air Force at this preliminary stage. If the software contains a few critical routines, however, the Program Office may want to participate as part of the reviewing team. A report summarizing the results of the reviews, recommendations, action items assigned, and their resolutions should be written and may be available for review by the Program Office via the Data Accession List.

The testing documentation as well as the actual code should be reviewed by the contractor's QA organization. A checklist for this review is given in Table 4-3. If the contractor maintains UDF's, the contractor's QA organization should sign the cover sheet, certifying that the testing meets the program's standards.

The Program Office will probably not have the resources to thoroughly review this documentation. Selected routines may be chosen for review. A check of some UDF's may be made to insure that the contractor's QA organization has reviewed and approved them.

#### 4.2.2 Software Integration Tests

After routines have successfully completed unit testing, they are combined and executed in integration tests to

- integrate routines into CPC's and CPCI's.
- verify CPC's and CPCI's through the anticipated range of operating conditions.
- verify requirements allocated to CPCI's which cannot be verified at the unit level.

**Table 4-3. Unit Testing Review Checklist**

- **Is the design clear? Does it do what is intended?**
- **Is the coding clear? Did you have trouble understanding it?**
- **Are the comments helpful in understanding the routine?**
- **Would you have trouble modifying it?**
- **Would you be proud of this work if it were yours?**
- **Does the code meet the program's coding standards?**
- **Do the tests meet the program's standards for unit testing?**
- **Does input data vary over allowable values including maximum, minimum, and nominal values? (All alike data, especially all zeros, is usually a poor choice).**
- **Is erroneous input data used? (This should be done. In fact all error conditions should be checked). Can you think of erroneous data conditions which were not used?**
- **Do the tests show that the routine has the functional capabilities allocated to it?**
- **Do the tests demonstrate that the code completely satisfies each requirement allocated to it?**
- **Does the actual output match the expected output?**

In a typical program, the responsibility for the software integration testing will be divided between the development and testing organizations using the guidelines given in paragraph 3.5. The testing activities are divided into testing levels, each of which integrates larger amounts of code until each CPCI has been integrated. The lowest levels of testing may be conducted by the programming team that developed the software. This testing is conducted and controlled in the same way as unit tests. Integration of software developed by more than one programming team requires more formality in the test team organization, documentation, and control of the software products.

The tests not conducted by the programming teams are executed by individual test teams, each consisting of a Test Manager, Test Conductors and a Test Librarian. The role of each is as follows:

- Test Manager - The director of the testing effort, this person is responsible for achieving the objectives of the test plan, and for the technical adequacy and completeness of the tests. This person establishes the detailed schedule, manages the resources and budget of the team, negotiates to insure that needed facilities and resources are available, and monitors and reviews the work of the members of the test team.
- Test Conductor(s) - These people write the test plans and procedures, execute the tests, analyze the results, and maintain the testing documents. Test conductors may have the primary or secondary responsibility for one or more tests and the responsibility may include the development of test tools and drivers. The test conductor aids in documenting software problems, resolving them and generating patches. To do this, the test conductor should have or develop a good working relationship with the software developers. (The software developers are responsible for resolving software problems).
- Test Librarian - This person maintains the software tools and products necessary to perform the testing. The Configuration Management Office (CMO) may, however, maintain some or all of these software tools and products. Most likely, though, CMO will not control all the items needed for software testing. The Test Librarian may submit some or all computer runs, generate object code for some or all of the software products, and maintain and control the tools needed by the test team. The Test Librarian may check to insure that the testing documentation is kept up to date.

Tests not performed by the programming teams require more formal documentation. Test plans and procedures should be written before the testing begins and should be published for review. The test plan establishes the criteria, methodology, responsibilities, and overall planning for the testing activity. It should contain the following information:

- Definition of each test
- Specific objectives of each test
- Location of each test
- Methods for preparation of input data
- Data recording requirements
- General procedures for data reduction
- Qualified personnel, numbers, responsibilities, and required knowledge and skills
- Requirements and procedures for controlling and documenting the test programs
- Test tools and simulations needed
- Schedule for each test and for development of each test tool and simulation
- Resources and facilities required

As with unit testing, the definition of the test cases is critical. At these higher testing levels, test cases are not generally defined by analyzing the logic of the individual routines. They are defined by considering the code as a black box, and using the Part 1 CPCI Development Specification to define inputs and expected outputs. Strategies used in defining these test cases include

- Partitioning - splitting input into equivalence classes and selecting representative input from each equivalence class.
- Boundary values - selecting input values at the boundary of the range of acceptable values, and picking input values that produce output at the boundaries of the output space.



- Cause and effect - creating a graphical representation of the code and selecting input to exercise each branch of the graph.
- Error exposure - using a combination of intuition, experience, and serendipity to select input most likely to uncover errors in the software.

A test procedure should be prepared for each test defined in the test plan. The procedure may be incorporated in the test plan or may be written later, incorporating comments from reviews of the test plan. In either event, it should be written before the test execution runs are made. The test procedures should including the following information

- Test objectives
- Location and schedule of test briefings, debriefings, and associated data reduction/analysis
- Reference to applicable test plans, specifications, manuals, and handbooks
- Requirements and responsibilities for console operators, test director, test conductors, test librarian
- Requirements for computer programs
- Test operation procedures
- A description of each test to be performed including test inputs, expected test outputs, expected results, requirements to be verified, methods of verification including inspection, analysis, demonstration, and formal test
- Requirements and procedures for recording, reduction, and analysis of test data

Both the test plan and procedure should be reviewed and approved before the testing begins.

The contractor may conduct an informal Test Readiness Review (TRR) before this testing begins to

- assess adequacy of the unit testing conducted.
- assess adequacy of planned testing.

- determine readiness to begin testing by reviewing the status of the code, tools, facilities, personnel, and CM procedure.

The Program Office may require the contractor to conduct a TRR before certain tests by including a requirement to do so in the Statement of Work.

To insure adequate documentation for each test case and to insure visibility, the contractor may establish a Test Development Folder (TDF) for each test. The TDF is an informal document internal to the contractor, which is generated and maintained during the development and execution of each test case run. It serves as a collection point for all information associated with that particular test case. Each TDF should contain the following information

- The test procedure to be followed during test case execution. If modifications are made during execution, the procedures are annotated to represent the as-run operations. The annotated procedure is initiated at each correction, as required, by the test conductor.
- Requirements tested and the procedure step that verifies each requirement.
- Hard copy output from the final test execution.
- Copies of any discrepancy reports generated as a result of a test execution and their final disposition.
- Test results/analysis reports.

A sample cover sheet from a TDF is shown in Figure 4-2. The cover sheet indicates that the Test Manager and a member of the contractor's QA organization may review this documentation to insure the technical adequacy of the testing conducted.

To insure that the test runs are repeatable, the contractor may use an internal form, a Test Execution Record (TER), to record the hardware and software configurations under which tests have been executed. Depending on the formality of the test, a TER may be written

TEST ID \_\_\_\_\_

SECTION NO.	DESCRIPTION	DUE DATE			DATE COMPLETE			TEST CONDUCTOR			DATE REVIEWED					
		D1	D2	D3	D1	D2	D3	D1	D2	D3	TM	QA	TM	QA	TM	QA
1	APPLICABLE DELIVERY DATES															
2	REQUIREMENTS															
3	TEST PROCEDURE															
4	REQUIREMENTS WORKSHEET															
5	TEST INPUTS															
6	EXPECTED RESULTS															
7	TEST EXECUTION OUTPUTS															
8	BENCHMARK TESTS															
9	TEST EXECUTION RECORDS															
10	DISCREPANCY REPORTS															
	TURNOVER TESTS															

Figure 4-2. Sample TDF Cover Sheet

for each test case run or only for successful runs. The TER should be originated by the Test Conductor, and reviewed by the Test Manager. Each TER should contain the following information

- Date and Time of Run
- Run ID
- Testing Level
- Hardware Configuration (including patches)
- Software Configuration (including patches)
- Data Base
- Test Support Hardware and Software
- Result of the Run
- Name and Signature of Test Conductor, Test Manager and Reviewer

Some of this information may be generated automatically with the hard copy output from the execution run.

The documentation produced by this informal testing is available for review by the Program Office via the Data Accession List. Since the guidelines for reviewing the information for informal and formal testing are the same, this topic is deferred to paragraphs 4.3 and 4.4.

#### 4.2.3 Preliminary Qualification Tests

The Preliminary Qualification Tests (PQT's) are a sequence of incremental tests which provide visibility and control to the Program Office of the computer program development between the CDR and FQT. These tests are conducted for selected complex or critical functions. Functions considered critical include those with

- time critical requirements
- performance critical requirements
- critical safety requirements

Special tests may be conducted for the PQT's, or some software integration tests may be designated as PQT's. In either case, the tests should be conducted by a team independent from the software developers. The team is organized as stated in paragraph 4.2.2.

The contractor writes a test plan for each CPCI in accordance with DI-T-3703A or DI-T-30715.

- The CPC's or functions to be tested during PQT
- The sequence of tests
- Performance and design requirements to be tested

Specifically, each test plan should contain the information detailed in paragraph 4.2.2. A test procedure containing the information in paragraph 4.2.2 is also written for each function or CPC to be tested. The tests are conducted using the guidelines given in paragraph 4.2.2.

Each CPCI test plan is reviewed by the Program Office. Preliminary CPCI test plans are reviewed at the PDR; the final CPCI test plan is reviewed at the CDR for the CPCI under review. To aid in these reviews, a checklist is given in Table 4-4.

The test procedures for the CPCI under consideration are reviewed at the CDR. These documents are very detailed, so a thorough review of each procedure may not be possible. A checklist for the review is provided in Table 4-5. These documents are not baselined at the CDR; the contractor updates the test procedures as the testing is conducted to reflect the testing as performed.

Test reports for the PQT's are written by the contractor and reviewed when available. The test reports may be a set of completed test procedures or one or more separate documents for each PQT. They should include the following information for each test

- Objective of the test
- Description of the test
- Summary of results of the test and an analysis of their significance

**Table 4-4. CPCI Test Plan Review Checklist**

- **Is each test defined and discussed separately? Are the testing approaches discussed?**
- **Are the testing objectives given? Are priorities listed?**
- **Is the scope of testing defined? Are limitations listed?**
- **Is the testing environment defined for each test? This should include**
  - **Drivers and/or stubs including emulators, ICS, and simulators**
  - **Data base value generators**
  - **Data logging tools**
  - **Instrumentation or patches required**
  - **Post processors and data reduction programs**
  - **The version of all controlled software products to be used**
  - **Hardware configuration including host computer(s) and needed peripheral equipment. If standard configurations are maintained by the Contractor or subcontractor, the appropriate configuration can be identified without listing the equipment used.**
- **Are required supporting items listed? These include**
  - **GFE and CFE**
  - **Computer resources**
  - **Support from other organizations including contractor and Air Force support**
  - **Logistic support**

**Table 4-4. CPCI Test Plan Review Checklist  
(Concluded)**

- **Are the organizations responsible for this testing activity identified? This includes responsibilities for**
  - **Maintaining hardware**
  - **Executing tests and analyzing results**
  - **Repairing errors and generating patches as needed**
- **Is a schedule given for each testing activity?**
- **Is the contractor's review procedure given? Are evaluation criteria stated for each test?**
- **Are test approval mechanisms stated including, when appropriate, review and approval of tests by**
  - **Quality Assurance**
  - **Test Manager**
  - **Other Contractor Reviewers and Supervisors**
  - **Program Office**

**Table 4-5. CPCI Test Procedure Review Checklist**

- **Is the purpose, organization, and structure of each procedure discussed?**
- **Is the change control mechanism identified? The CM plan may be referenced for controlled items. For non-controlled items which require some internal control, for example a post processor, a change control mechanism should be given.**
- **Are procedures given for recording the testing progress, and for documenting and resolving problems?**
- **Is the procedure complete? Do you know how to execute each test after reading the procedure? Can you tell the difference between satisfactory and unsatisfactory results?**
- **Is the expected output listed for each test?**
- **Are the expected output values correct for the input values? Does it agree with the Part 1 CPCI Development Specification? Does it surprise you? (If so, there may be a problem with the requirement).**
- **Are guidelines given to aid in interpreting the output?**



- The hardware and software configuration used including a list of all software patches (may be included by referencing the test procedure).
- A discussion of both outstanding and resolved problems, their status and corrective action, including requirements the software does not satisfy, and actual output which doesn't agree with the expected output.

Testing meetings attended by representatives from the contractor's software development, test, QA and CM organizations, and, depending on the project, the Program Office, may be held by the contractor. These meetings are held to

- monitor the progress of the testing.
- identify problems early.
- coordinate testing among the organizations involved.

Attendance at these meetings is an excellent way to keep informed of the testing activity.

#### **4.2.4 Formal Qualification Test**

The Formal Qualification Test (FQT) is a comprehensive test of a CPCI conducted by the contractor after the software has been developed. It is designed to qualify the software for system integration and later operational testing. The testing should exercise every function of the CPCI regardless of previous PQT's. The PQT's may evolve into the FQT, but should not be substituted for any part of the FQT. When practical, the FQT is conducted at the developer's facility. Testing of a CPCI which is sensitive to the operational environment may be conducted at the system's testing location.

Much of the information given in paragraph 4.2.3 for PQT's applies here. The test team is organized in the same way. The test plan and procedures are reviewed by the Program Office before the testing begins. The documents are controlled by the contractor.

The test plans and procedures are reviewed using the guidelines in paragraph 4.2.3. Also, a TRR may be held before the FQT is conducted.

#### **4.3 RETESTING AND MODIFICATIONS TO SUPPORT SYSTEM TESTING**

After the software is baselined, changes will be made to resolve problems found in testing, to add features and enhancements to the software, and to change capabilities as a result of changes to the specifications. Often changes are made to the software as a result of system testing. It is frequently more cost effective to modify the software to resolve problems than to change hardware.

Each change does not result in a new software product. Changes are accumulated and the affected software products are retested and periodically rebaselined by the contractor's CMO as new versions.

Each revised software product must be retested to

- verify that changes resolve problems.
- verify that changes add the features, enhancements or capabilities intended.
- insure that changes do not introduce new problems (the ripple effect).

Each problem encountered should be documented in a DR. This provides an efficient way to gather statistics on errors and software failures, which can be used in assessing the quality and reliability of the software that is produced.

Retesting, sometimes called regression testing, begins with reexecuting the unit tests for each modified routine. The tester should reexamine the tests defined for the routine, and modify them to insure that they meet the program's standards for unit testing. Input and expected output should be recomputed before the unit tests are executed.

Some integration tests may be rerun, depending on the testing level and the extensiveness of the changes. Integration tests run by the programming teams may be rerun each time the computer program is changed. Those performed by other organizations may be rerun only occasionally. Extensive changes may require rerunning all the previous tests and executing new ones. As with the unit testing, the tester should analyze the revised computer program, and modify and add tests using the guidelines given in paragraph 4.2.2 for defining test cases. Input and expected output must be defined before these tests are executed.

During the early testing, existing test teams may have the responsibility to retest the code. After this testing has been completed and the test teams have been disbanded, a new team is usually formed to retest the computer program. This team is structured as stated in paragraph 4.2.2. It defines test cases and executes them using the guidelines given there. The team may define a standard set of test cases, benchmark tests, which will be executed for each revision of the computer program.

To aid in this activity test scripts may be prepared. The test scripts are test tools which provide an automated way to execute the benchmark test cases, summarize the results, compare the output of the computer run with previous results, and print the differences. This reduces the chance of overlooking changes in the performance of the software.

The FQT procedures are maintained and used throughout the operational life of the software. The procedures are revised as needed to test any revised code. The tests are then rerun under the same controlled conditions as the original FQT.

The computer program products are controlled as discussed in paragraph 3.3.3. All the testing materials including the test scripts and post processors may also be controlled.

The releasing contractor prepares a Version Description Document (VDD), in accordance with DI-E-3121, to accompany the release of each version of a CPCI. The contractor must also prepare a VDD, identifying the item delivered and its changes, for each release of an interim change to a CPCI.

APPENDIX A  
ANNOTATED BIBLIOGRAPHY

Anderson, T., and S.K. Shrivastava, "Reliable Software: A Selective Annotated Bibliography," Software - Practice and Experience, 8, 59-76, 1978.

Sixty-four references to papers, books, and conference proceedings on software reliability are selected and annotated. The papers selected are generally recent and introductory to provide a current survey of the subject.

Boehm, B.W., R.K. McClean, and D.B. Urfrig, "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," IEEE Trans. on Software Engineering, SE-1, 1, 125-133, March 1975.

The paper discusses types of software errors encountered on large software projects. A taxonomy of the causes of software errors and an analysis of design error data are discussed. A prototype of an automated aid to detect inconsistencies between assertions of input and output of elements of a software design is presented.

Boehm, B.W., "Software Engineering," TRW Software Series, TRW-SS-76-08, 1976.

This paper is a survey of the state of the art of software engineering and likely future trends. The survey discusses requirements engineering, design, coding test, and maintenance. The domain of applicability of techniques rather than details of their workings is presented. An extensive bibliography is included.

Buckley, Fletcher, "A Standard for Software Quality Assurance Plans," Computer, 12, 8, 43-50, August 1979.

The paper presents a draft "Standard for Software Quality Assurance Plan," the work of the Software Engineering Standards Subcommittee of the IEEE Computer Society's Technical Committee of Software Engineering.

DeMillo, Richard A., Richard J. Lipton, and Frederick G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," Computer, 11, 4, 34-41, April 1978.

The paper discusses practical strategies for selecting test cases, exploiting the fact that programs are close to being correct. Test cases should be simply defined by using a detailed knowledge of the code and its application. The coupling effect is discussed. Simple test cases, those that distinguish programs differing from correct ones by only simple errors, are so sensitive that they also distinguish complex errors.

Duke, M. O., "Testing in a Complex Systems Environment," IBM Systems Journal, 4, 353-365, 1975.

This paper is a discussion of the testing performed on a complex computer system, Information Management System/Virtual Storage (IMS/VS). Two types of testing, functional and performance testing, are discussed. The testing methodology, execution, and post-test analysis used on this program are summarized. The need for test tools, libraries, and editors is emphasized.

Fairley, Richard E., "Tutorial: Static Analysis and Dynamic Testing of Computer Software," Computer, 11, 4, 14-23, April 1978.

Two complementary approaches to software testing, static analysis and dynamic testing, are discussed. Static analysis involves obtaining global information about the structure of the program, while dynamic testing investigates the program's run time behavior. Bottom up, top down, and mixed testing strategies are discussed. A software system design to obtain data on the execution of batch-mode ALGOL 60 source programs is discussed.

Finfer, Marcia, Jon Fellos, and Dan Casey (System Development Corporation), "Software Debugging Methodology," RADC-TR-79-57, Vol. 1 (of three) Final Technical Report, April 1979.

The document surveys research currently being conducted in software debugging at the integration level. Emphasis is given to assessing tools and techniques used for embedded software. The paper was written to present a software debugging methodology applicable to diverse environments.

Gerhart, Susan L., and Laurence Yelowitz, "Observations of Fallibility in Applications of Modern Programming Methodology," IEEE Trans. on Software Engineering, SE-2, 3, 195-206, September 1979.

The paper defines three types of errors: specification errors, systematic construction errors, and proved program errors. It considers twelve algorithms, many of which are used as examples in teaching modern programming practices. The paper shows that even those programs contain examples of each of the above types of errors. Software errors abound!

Gilb, Tom, Software Metrics, Weinthrop Publisher, Inc., Cambridge, Mass., 1977.

The book presents techniques for measuring many software characteristics such as reliability, flexibility, structuredness, performance, and resource needs. Practical applications of these techniques are given. Helpful checklists are given in the appendices, including a checklist for inspection of COBOL programs, and a description of the process of test inspections, with some sample forms.

Glass, Robert L., Software Reliability Guidebook, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1979.

A survey of technological and managerial techniques used to develop reliable software is presented as a menu. Technological techniques discussed include requirements specification, design, implementation, checkout, and maintenance techniques. Managerial topics of planning, organization, documentation, production, and scheduling are discussed. References are given for each item introduced. The techniques presented are ranked in order of importance and recommendations for the use of each methodology are given for projects as functions of cost and importance of reliability.

Hetzel, W.C., ed., Program Test Methods, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1973.

The book contains a series of papers on various aspects of software testing presented at the Computer Program Test Methods Symposium held at the University of North Carolina, Chapel Hills, North Carolina, June 21-23, 1972. The book also has an extensive annotated bibliography.

Ingrassia, F.S., "The Unit Development Folder (UDF): An Effective Management Tool for Software Development," TRW Software Series, TRW-SS-76-11, October 1976.

The paper discusses the concept and use of the UDF in software development and unit testing.

Mills, Harlan D., "Software Development," IEEE Trans. on Software Engineering, SE-2, 4, 265-273, December 1976.

The paper states that software development is a critical factor in the use of automatic data processing. Improvement will occur only when the software design and methodology improves. Reliability must be put into the design. Suggestions to do this are given. The paper also discusses the error day - one software error existing for one day is one error day.

Mullin, F.J., "Considerations for a Successful Software Test Program," TRW Software Series, TRW-SS-77-01, January 1977.

This paper discusses some managerial considerations for software testing. It emphasizes the need for early test planning and describes the activities and responsibilities of the group assigned to formally test the software.

Myers, Glenford J., Software Reliability: Principles and Practices, J.Wiley and Sons, NY, NY, 1976.

The book covers the elements of software reliability. Design techniques and methods are covered in detail. Software testing including basic principles, module, function and system testing, and debugging, are presented. (Much of the material on software testing also appears in the author's book, The Art of Software Testing.) Other topics such as programming languages, computer architectures, and reliability models are given. References, checklists, and many examples are presented. This book, as the one below, is very readable.

Myers, Glenford J., The Art of Software Testing, J. Wiley and Sons, NY, NY, 1979.

The book is a practical rather than theoretical discussion of the nature of software testing. The book presents and explains techniques of test case definition or testing design at the different levels of testing. Program inspections, walkthroughs, and reviews are discussed, with detailed checklists given to aid in these activities. Top down and bottom up testing methods are analyzed. Debugging methods and test tools are discussed in detail. The book is an excellent statement of the state of the art in software testing today and is well worth reading.

Ramamoorthy, C.V., and S.B.F. Ho, "Testing Large Software with Automated Software Evaluation Systems," IEEE Trans. on Software Engineering, SE-1, 1, 46-58, 1975.

The article surveys automated tools used in software design and testing. The tools are classified into six functional areas and the characteristics of each are discussed. Tools used in industry are described. The paper contains a list of 31 references on the subject.

Reifer, Donald J., (The Aerospace Corporation), "Microprogram Verification and Validation," SAMSO-TR-76, 217, Interim Report, February 1976.

This report documents present methods and techniques used for verification and validation of microprograms. Six methods (diagnostics, test and evaluation, program proving, simulation/emulation, graph-theoretic, and monitors) are discussed.

Reifer, Donald J., and Stephen Trattner, "A Glossary of Software Tools and Techniques," Computer, 10, 7, 52-60, July 1977.

This paper describes 70 types of tools and techniques used in the software life cycle. Some of the information in this paper can be found in Appendix B of the SAE Guidebook on Verification, Validation and Certification.



Tausworthe, Robert C., *Standardized Development of Computer Software, Part II, Standards*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.

The book details the standards used by JPL to develop software to upgrade the digital data systems of all of JPL's deep-space stations. Standards are given for software requirements, program design, coding, testing, quality assurance, and documentation. Detailed examples are given. Eleven appendices are used to present detailed outlines for specifications, notebooks, and manuals, and to illustrate sample forms for product control and status reporting.

Zelkowitz, Marvin V., "Perspectives on Software Engineering," *Computing Surveys*, 10, 2, 197-216, June 1978.

This paper summarizes the state of the art of software engineering. The software development life cycle, requirements analysis, specification, design, coding, testing, and operation and maintenance are discussed. Both management and programming considerations in software development are presented. Topics discussed include the use of librarians, chief programming teams, estimation techniques, design walkthroughs, automated tools, formal testing, structured programming, and design techniques including top down design and development, virtual machines, and program design language (PDL).